

20

QUE LE FORTH SOIT AVEC VOUS

JANVIER 1986

le FORTH soit avec Vous

EDITORIAL

Et voilà! Une année s'achève, une autre commence. Ainsi en est-il aussi de JEDI. Mille neuf cent quatre vingt cinq nous aura prouvé votre intérêt pour ce magazine d'un autre type. Cette nouvelle année, nous voulons aller encore plus loin, vous proposer de plus en plus d'outils et de moyens pour mieux appréhender le fantastique avenir que nous promet l'informatique. Pour ce faire, nous avons besoin de vous, de vos idées, de vos articles, de votre bonne volonté. Nous sommes sûrs de pouvoir compter sur vous. Aussi, de tout coeur, nous souhaitons-vous une bonne et heureuse année!

M.ROUSSEAU M.PETREMANN

SOMMAIRE



ASSOCIATION JEDI 8, rue Poirier de Narçay 75014 PARIS
Tel: (1) 45.42.88.90 (de 10h à 18h)

UNE REGLE A CALCULS POUR FORTH

Un article de Nathaniel GROSSMAN,
Dpt Mathematics UCLA LosAngeles CA 90024
traduit par Christophe LAVARENNE.

Si la plupart des utilisateurs d'arithmétique en virgule flottante ne ressentent pas toujours le besoin d'une extension complète en virgule flottante, ils peuvent trouver des applications pour une "règle à calculs". De petite taille, une telle extension permettrait le calcul, en virgule flottante simple précision, des fonctions mathématiques usuelles, y compris addition et soustraction. Comme avec une règle à calculs, le domaine de validité des fonctions serait limité, laissant à l'utilisateur le soin des calculs d'échelle. Voici une implémentation de règle à calculs pour Forth, commençant par les quatres opérations de base implémentées par Martin Tracy, les autres fonctions mathématiques étant dérivées de l'algorithme CORDIC unifié.

Du temps de la plume, du papier et des tables numériques, la règle à calculs facilita les travaux de l'Homme. Cet instrument, qu'on pourrait présenter comme un ancêtre, non miniaturisé, d'une puce interactive intégrant les fonctions arithmétiques du calcul flottant, a permis d'économiser un nombre considérable d'heures qui auraient été consacrées à de longues divisions et à d'incessantes utilisations de tables numériques. Quand les calculateurs mécaniques, puis électroniques devinrent disponibles, puis courants, les calculs arithmétiques à grande précision ne devinrent pas plus difficiles que ceux à faible précision, ces derniers se résumant simplement à arrondir ou à tronquer les résultats en grande précision.
Pour le concepteur de matériel ou de logiciel, le passage de faible à grande précision n'est pas une étape triviale. L'utilisateur, pour qui l'implémentation est souvent transparente, peut se plaindre d'avoir à supporter le temps nécessaire à un calcul en haute précision pour obtenir un résultat en faible précision. De même, lors du transport d'une application ne nécessitant que du calcul en faible précision, les efforts nécessaires au transport des outils de calcul en haute précision semblent disproportionnés.
Cet article présente une implémentation genre règle à calcul de l'algorithme de CORDIC permettant à lui seul de générer les fonctions mathématiques usuelles, y compris multiplication et division si nécessaire. Un algorithme unifié est des plus attirant pour un concepteur de matériel: les calculatrices de poche Hewlett-Packard sont équipées d'un processeur CORDIC câblé [1,6]. Pour le concepteur d'outils de calcul en virgule flottante, CORDIC est également intéressant. En général, pour des raisons de vitesse d'exécution, CORDIC est souvent implémenté en virgule fixe, ce qui réclame des mises à l'échelle avant et après exécution. Les opérations de mise à l'échelle sont reportées dans une routine en soi très simple. Ceux qui ont utilisé une règle à calculs se rappelleront que les mises à l'échelle étaient laissées entièrement aux soins de l'utilisateur par calcul mental. L'implémentation présentée ici se situe à mi-chemin, sacrifiant la vitesse d'exécution au profit d'un calcul en virgule flottante simplifiant les problèmes de mise à l'échelle. Elle est tirée d'une extension compacte en simple précision que Martin Tracy a écrite et baptisée "Maths Zen" pour rire, aussi avons nous baptisé "Règle à calucls Zen" les écrans que nous y avons ajouté. La règle à calculs Zen, comme la "vraie", laisse à l'utilisateur le soin de vérifier le domaine d'utilisation des différentes fonctions.
Le reste de cet article décrit l'algorithme CORDIC puis commente les écrans présentant notre implémentation. Nous n'entrerons pas ici dans les détails de démonstration du fonctionnement de CORDIC: ceux qui s'y intéressent pourront se reporter à la littérature existante. L'introduction simple de Schelin [5] est sans doute le meilleur point de départ. Schelin donne une longue bibliographie sur CORDIC et d'autres sujets s'y rapportant ; entre autres, les articles de Schmid et Bogacki [6], Volder (le concepteur de CORDIC) [8], et Walther [9] ont été les plus utilisés pour la préparation de ce document. Les articles de Furman [3] et de Freese [4] dans "Forth Dimensions" contiennent une implémentation CORDIC en virgule fixe pour les fonctions Sinus et Cosinus en Forth. Nous avons intégré pratiquement toutes les fonctions élémentaires usuelles citées par Duncan et Tracy [2] dans leur proposition de standard en virgule flottante.

L'ALGORITHME CORDIC.

Comme le fait remarquer Schelin [5], l'algorithme CORDIC est remarquable par le fait qu'il ne fait pas appel aux techniques du calcul différentiel ou intégral pour calculer les valeurs des fonctions élémentaires. Etant données trois valeurs initiales, appelées soit x_0, y_0, z_0 soit x_1, y_1, z_1 suivant la fonction à calculer, des séquences de triplets x_k, y_k, z_k sont générées suivant le shéma de récurrence suivant:

$$x_{k+1} = x_k - m * delta_k * y_k * 2^{-k}$$
$$y_{k+1} = y_k + delta_k * x_k * 2^{-k}$$
$$z_{k+1} = z_k - delta_k * epsilon_k$$

Partant de k=0 ou 1 comme approprié, les itérations doivent être poursuivies jusqu'à k=n. Le paramètre m (appelé le mode) vaut +1 ou -1. Les nombres $epsilon_k$ se présentent sous deux aspects, présents dans les tableaux +EPS et -EPS:

+EPS: $epsilon_k=\tan^{-1}(2^{-k})$, pour k=0,1,...,n

-EPS: $epsilon_k=\tanh^{-1}(2^{-k})$, pour k=1,...,n

Dans notre implémentation, n=14. Les nombres $delta_k$ sont choisis égaux soit à +1 soit à -1 de manière à forcer soit y_k soit z_k vers zéro. Si z_k est forcé vers zéro, le processus est appelé rotation, sinon si y_k est forcé vers zéro il est appelé vectorisation. Une obscure condition technique sur les epsilons exige que les itérations pour k=4 et k=13 soient répétées et k=0 soit omise, pour la vectorisation seulement. Enfin, deux constantes sont nécessaires à l'initialisation des récurrences:

$$1/K = (1+4^{-0})^{1/2} * (1+4^{-1})^{1/2} * \ldots * (1+4^{-14})^{1/2} = 0.6073$$

$$1/K' = (1-4^{-1})^{1/2} * (1-4^{-2})^{1/2} * \ldots \langle 4 \text{ répété} \rangle \ldots \langle 13 \text{ répété} \rangle * (1-4^{-14})^{1/2} = 1.2076$$

Les instructions pour évaluer les fonctions spécifiques sont données dans le tableau ci-dessous, qui est adapté de celui donné par Schelin [5]. Le cas m=0 qui peut être utilisé pour effectuer les multiplications et les divisions est omis ici parce qu'il n'est pas utile quand +, -, * et / sont déjà implémentés en virgule flottante.

Le shéma CORDIC en binaire

Rotation (z_k->0)

$delta_k$=+1 si $0<z_k$, -1 si $z_k<0$

m=+1: $x_0=1/K$, $y_0=0$, $z_0=t$ *donnent* $x_{n+1}=\cos(t)$, $y_{n+1}=\sin(t)$

m=-1: $x_0=1/K'$, $y_0=0$, $z_0=t$ *donnent* $x_{n+1}=\cosh(t)$, $y_{n+1}=\sinh(t)$

$e^t=x_{n+1}+y_{n+1}$

Vectorisation (y_k->0)

$delta_k$=+1 si $y_k<0$, -1 si $0<y_k$

m=+1: x_1 donnée, y_1 donnée, z1=0 *donnent* $z_{n+1}=\tan^{-1}(y_1/x_1)$

$x_{n+1}=K*(x_1{}^2+y_1{}^2)^{1/2}$

m=-1: x_1 donnée, y_1 donnée, z1=0 *donnent* $z_{n+1}=\tanh^{-1}(y_1/x_1)$

$x_{n+1}=K'*(x_1{}^2+y_1{}^2)^{1/2}$

($x_1=t+1$ et $y_1=t-1$ *donnent* $\ln(t)=2*z_{n+1}$)

($x_1=t+1/4$ et $y_1=t-1/4$ *donnent* $t^{1/2}=x_{n+1}/K'$)

LES ECRANS FORTH

Nous supposons qu'une implémentation de +, -, * et / est disponible en virgule flottante, accompagnée des utilitaires nécessaires aux entrées-sorties (conversion ascii-binaire). Les nombres sont supposés être manipulés dans la même pile que les entiers avec la caractéristique (l'exposant en base 10) sur un mot au sommet de la pile, et la mantisse juste en-dessous, sur un nombre non prescrit de mots. La caractéristique, comme la mantisse doivent être des entiers décimaux signés (le nombre d'itérations pour la boucle DO...LOOP la plus interne pourrait être réduit considérablement si le calcul se faisait en virgule flottante binaire, c'est-à-dire avec un exposant en base deux).

Les écrans n°6 à 15 contiennent la "règle à calculs Zen". Seul l'écran 6 y est dépendant de l'implémentation, et l'implémentation des quatres fonctions arithmétiques en virgule flottante est celle de Martin Tracy "maths Zen" [7], qui utilise une mantisse en simple précision, occupant donc deux mots avec la caractéristique. Du coup, la plupart des fonctions de manipulation de la pile ne sont que des synonymes des opérateurs double-précison (2DUP, 2! ...). Les lignes 4, 5 et 6 de l'écran n°6 sont au cœur du "truc" pour diviser par 2. Si le but était de diviser par 10, le plus simple serait d'utiliser 1- pour retrancher un de la caractéristique du nombre flottant sur la pile. De même, 1- suffirait à diviser le nombre flottant par 2 si les nombre étaient représentés avec une charactéristique en puissances de 2. Ici, comme nous l'avons mentionné, nous travaillons avec un système hybride, exécutant CORDIC en virgule flottante décimale. Nous devons donc payer le désavantage qui consiste à effectuer une division par 2 en multipliant d'abord par 5 puis en divisant par 10. Pour qu'aucun chiffre significatif ne soit perdu par débordement ("overflow") lors de la multiplication, la mantisse en simple précision est tout d'abord étendue en double précision (2 mots), puis le résultat du produit est reconverti en simple précision par la fonction TRIM de Tracy. Le mot F2^N/ (r n --- r/2^n) divise le réel r par l'entier 2^n et doit effectuer les divisions par 2 nécessaires au moyen d'une boucle DO...LOOP (en virgule flottante binaire, on pourrait tout simplement écrire : F2^N/ (r n --- r/2^n) - ;). Le mot FLOAT aux lignes 12 et 14 est le discriminateur de Tracy qui transforme en réel ce qui pourrait être pris pour un entier double : 3.1416 FLOAT laisse 31416 -4 sur la pile. Enfin, FF est un synonyme servant d'abbréviation pour FLOAT.

L'écran n°7 contient un assortiment de constantes et de variables utiles. Si la précision devait changer, seules les valeurs de 1/K et de 1/K' seraient à changer. Ces deux constantes pourraient être calculées avec plus de précision par les formules données ci-dessus, en ajoutant au produit les facteurs suivants, jusqu'à ce que la puissance de 4 soit nulle pour la précision recherchée (pour 1/K', l'itération suivante à répéter se trouve à k=40). Les trois variables flottantes, XX, YY et ZZ et la variable entière NDX résultent d'un choix délibéré de l'auteur pour améliorer la lisibilité du code. Les opérations d'accès mémoire en sont un peu moins efficaces, mais de toute façon, le plus gros du temps d'exécution se situe au niveau de l'addition en virgule flottante. Il sera facile de remplacer les manipulations de variables par des manipulation au niveau de la pile, mais celles-ci risquent de devenir prohibitives pour des précisions de mantisse plus importantes.

Le mot FARRAY crée une table de constantes réelles flottantes et empile à l'exécution celle sélectionnée. Il est utilisé pour générer les tables +EPS et -EPS. Le $k^{ième}$ élément de +EPS vaut $\tan^{-1}(2^{-k})$, et celui de -EPS vaut $\tanh^{-1}(2^{-k})$ (sauf pour k=0 pour lequel $\tanh^{-1}(1)$ est indéfini, et remplacé par une valeur nulle). Pour sélectionner plus librement une des deux tables pour la valeur des epsilons à utiliser, le choix passe par EPS, un mot différé (créé par DEFER).

Comme m et delta ne prennent que les valeurs +1 ou -1, la multiplication par l'un ou l'autre peut se résumer à un NEGATE conditionné. Les deux processus de rotation et de vectorisation définissent de manière un peu différente le choix des delta, d'où les deux mots R-DELTA et V-DELTA, et pour les accès aux variables, les mots R-STORE et V-STORE. L'unité de l'algorithme CORDIC fait reporter le choix entre les deux versions à travers le mot différé !STACK.

Les trois mots NEW_X, NEW_Y et NEW_Z supportent les itérations de la récurrence et représentent la partie principale des deux mots principaux, ROT'ING et VEC'ING qui forment le noyau, à travers le mot différé DO-IT, de la boucle centrale du mot CORDIC.

A l'écran n°11, les deux premiers mots décident du mode en plaçant un drapeau et en sélectionnant la table d'epsilons appropriée. Le mot CORDIC est au cœur de tous les calculs de fonctions. La séquence NDX ! DO-IT aurait pu être factorisée, mais a été laissée telle quelle pour plus de clarté.

Les écrans n°12 à 14 proposent une sélection généreuse de fonctions en virgule flottante: cos, sin, tan, atan (trigonométriques), p>r, r>p (transformations coordonnées rectangulaires-polaires), exp, cosh, sinh, tanh, atanh (hyperboliques), sqrt (racine carrée), ln (logarithme naturel). Ce qui représente l'ensemble des fonctions spécifiées par Duncan et Tracy [2], les autres pouvant facilement se déduire de celles-ci.

Ayant mentionné qu'une règle à calculs laisse à l'utilisateur la tâche de vérifier les domaines de validité des fonctions, nous donnons ceux-ci suivant Walther [9].

Domaines de convergence CORDIC

fonctions	(domaines)
sin, cos, tan	(-1.74, 1.74)
$\tan^{-1}$	(-infini, +infini)
sinh, cosh, tanh, exp	(-1.13, 1.13)
$\tanh^{-1}$	(-0.81, 0.81)
ln	(0.10, 9.58)
sqrt	(0.03, 2.42)

Enfin, notons que les résultats produits par la règle à calculs Zen seront précis à une ou deux unités à la troisième décimale, l'erreur étant dûe au manque de décimales de garde (c'est pour conserver les effets de tronquature que nous n'avons pas adopté la méthode de Schmid et Bogacki [6] qui réclamerait beaucoup plus d'additions flottantes pour obtenir la même précision). Le code donné tel quel est compilé en 1580 octets sur le MasterForth de Micromotion (une réalisation enrichie de Forth-83) sur Apple IIe. Maths Zen de Tracy réclame 500 octets supplémentaires, soit un total de 2080 octets.

REFERENCES

1. D. **Cochran**, Algorithms and accuracy in the HP35, HewlettPackard J. (Juin 1972, p10-11)
2. R. **Duncan** & M. **Tracy**, The FVG standard floating-point extension, DrDobb's Journal (Septembre 1984, p110-115)
3. D. **Freese**, CORDIC algorithm revisited, Forth Dimensions V, n°3 p24-25
4. A.T.**Furman**, The CORDIC algorithm for fixed-point polar geometry, Forth Dimensions IV, n°1 p14-15
5. C.W.**Schelin**, Calculator function approximation, American Mathematical Monthly (Mai 1983, p317-25)
6. H.**Schmid** & A.**Bogacki**, Use decimal CORDIC for generation of many transcendantal functions, Electrical Design News Magazine (Février 1973, p64-73)
7. M.**Tracy**, informal communication, 1984.
8. J.**Volder**, The CORDIC comuting technique, IREE Transactions On Computers (Septembre 1958, p330-334)
9. J.**Walther**, A unified algorithm for elementary functions, Joint Computer Conference Proceedings 38 (printemps 1971, p379-85)

FORTH LA VIRGULE FLOTTANTE EN 83-Standard

par Martin TRACY
LA CHAPTER
et Nathaniel GROSMAN
Dept. of Math. UCLA

Extension Standard en Virgule Flottante

1.0 Syntaxe d'un nombre en virgule flottante

Si l'extension a été greffée à un système programmé en 83-Standard, une chaîne de caractères de la forme suivante sera interprétée ou ou compilée en tant que nombre réel:

nnnn.nnnnExx

Le "E" sert à forcer la conversion en nombre réel. La présence des digits numériques avant ou après le "E" n'est pas requise par ces spécifications, mais est demandée dans de nombreuses implémentations. Le signe "-" doit précéder la mantisse et l'exposant, un "+" étant également permis à l'exposant. Le point décimal est optionnel et utilisé seulement dans la mantisse. Pour exemple, tous les nombres qui suivent sont valides:

```
-.0001E5
100.0E+0
1000.E-15
```

Le mot FNUMBER doit être utilisé par l'application pour convertir les chaînes en nombres réels. La base courante du système est basculée en décimal avant une entrée ou affichage d'un réel. Si ce n'est pas le cas, le nombre obtenu ne sera pas défini.

2.0 Affichage d'un nombre en virgule flottante

Deux opérations primitives d'affichage sont valables: F. qui ajuste la position du point décimal et affiche le nombre sans exposant, et E. qui affiche le nombre en notation avec exposant. Des formatages de sortie plus complexes sont disponibles dans divers systèmes et sont décrits dans le Glossaire des Mots Optionnels en Virgule Flottante.

3.0 La pile en virgule flottante

Les implémentations diffèrent en ce sens que certaines ont une pile en virgule flottante séparée de la pile de données, alors que d'autres partagent la même pile. Cette proposition d'extension n'expose pas les avantages ou inconvénients de l'une ou l'autre méthode. L'origine ou la destination d'un nombre réel sera toujours la pile des réels si une pile séparée existe.

4.0 Portabilité

Pour assurer la portabilité d'une application, il est important que le programmeur n'avantage pas sa conception particulière des nombres en virgule flottante. Les diverses versions d'opérations en accès mémoire et opérations de pile doivent s'appliquer à tous les cas de nombres réels, également dans le cas où ils coincident avec d'autres opérateurs FORTH. Par exemple, si le système supporte un format réel en 32 bits, FDUP et 2DUP auront le même effet --- mais ce sera FDUP qui sera utilisé afin de conserver la portabilité.

5.0 Glossaire Virgule Flottante

Le jeu de mots proposé en extension pour la virgule flottante est décrit dans un format de glossaire standard. Les symboles indiquent l'ordre dans lequel les paramètres d'entrée doivent être déposés sur la pile de données. Trois traits "---" marquent la partie exécution; les paramètres restant sur la pile sont indiqués. Dans cette notation, le sommet de la pile est à droite. Ces symboles sont: VF1

adr	adresse mémoire
b	octet huit bits (octet de poids fort ignoré)
r, r1	nombre virg. flottante (longueur dépendante de l'implémentation)
f	flag booléen, 0=faux, -1=vrai
n	entier 16 bits signé
u	entier 16 bits non signé
d	entier 32 bits signé
du	entier 32 bits non signé
I	immédiat
C	en compilation seulement

Opérateurs Arithmétiques Requis en Virgule Flottante

F+ r1 r2 --- r3

Addition en virgule flottante, délivre la somme de r1 et r2.

F- r1 r2 --- r3

Soustraction en virgule flottante, délivre la différence de r1 moins r2.

F* r1 r2 --- r3

Multiplication en virgule flottante, délivre le produit de r1 et r2.

F/ r1 r2 --- r3

Division en virgule flottante, délivre le quotient de r1 divisé par r2.

F** r1 r2 --- r3

Délivre la valeur de r1 élevée à la puissance r2.

FABS r --- |r|

Délivre la valeur absolue d'un nombre réel.

FNEGATE r --- -r

Change le signe d'un nombre réel.

FSQRT r1 --- r2

Extraction de la racine carrée d'un nombre réel.

FMAX r1 r2 --- rmax

Délivre le plus grand de r1 ou r2.

FMIN r1 r2 --- rmin

Délivre le plus petit de r1 ou r2.

Fonctions Transcendantes Requises en Virgule Flottante

FLOG r1 --- r2

Log en base 10 de r1.

FLN r1 --- r2

Log en base e de r1.

VF2

FALOG   r1 --- r2

Délivre la valeur de 10 à la puissance r1.

FALN   r1 --- r2

Délivre la valeur de e à la puissance r1.

FSIN   r1 --- r2

Délivre de sinus de r1. L'argument d'entrée est exprimé en radians.

FCOS   r1 --- r2

Délivre le cosinus de r1. L'argument d'entrée est exprimé en radians

FTAN   r1 --- r2

Délivre la tangeante de r1. La tangeante de 90 à 270 degrés renvoie le plus grand nombre réel représentable dans l'implantation.

FASIN   r1 --- r2

Arc-sinus, valide pour -1<=r1<=1. Le résultat est situé dans l'intervalle -¶/2 à ¶/2 radians.

FACOS   r1 --- r2

Arc-cosinus, valide pour -1<=r1<=1. Le résultat est situé dans l'intervalle 0 à ¶ radians.

FATAN   r1 --- r2

Arc-tangeante, valide pour tous les réels r1. Le résultat est situé dans l'intervalle -¶/2 à ¶/2 radians.

Opérateurs Logiques Requis en Virgule Flottante

F0=   r --- f

Vrai si le nombre en virgule flottante est égal à zéro. Le nombre réel est supprimé de la pile et remplacé par le flag qui est déposé au sommet de la pile de données.

F0<   r --- f

Vrai si le nombre réel est inférieur à zéro. Le nombre réel est supprimé de la pile et remplacé par le flag qui est déposé au sommet de la pile de données.

F=   r1 r2 --- f

Vrai si le réel r1 est égal au réel r2. Les nombres réels sont supprimés de la pile et remplacés par le flag qui est déposé au sommet de la pile de données.

F<   r1 r2 --- f1

Vrai si le nombre réel r1 est inférieur au réel r2. Les nombres réels sont supprimés de la pile et remplacés par le flag qui est déposé au sommet de la pile de données.

Operateurs de Manipulation de Pile Requis en Virgule Flottante

FDROP   r ---

Détruit le nombre réel situé au sommet de la pile de données.

VF3

FDUP   r --- r r

Duplique le réel situé au sommet de la pile.

FOVER   r1 r2 --- r1 r2 r1

Duplique le second réel sur le sommet de la pile.

FSWAP   r1 r2 --- r2 r1

Inverse la position des deux réels situés au sommet de la pile.

FROT   r1 r2 r3 --- r2 r3 r1

Dépose le troisième réel sur le sommet de la pile.

Manipulations Requises en Virgule Flottante

FLOAT   d --- r

Convertit un entier double précision en un nombre réel, et détruit l'entier double précision de la pile de données. Le nombre réel est déposé sur la pile (pile de données FORTH ou pile de réels).

FIX   r --- d

Convertit un nombre réel en entier double précision le plus approchant, détruisant le nombre réel de la pile et laissant le résultat sur la pile de données. L'approximation, lors de la conversion, est réalisée vers l'entier le plus proche. Si le réel est trop petit, le résultat est nul, un dépassement de capacité provoque une condition d'erreur. Exemple: 2.7E0 renvoie l'entier double précision 3.

INT   r --- d

Tronque un nombre réel en nombre double précision, supprimant le nombre réel et laissant le résultat sur la pile de données. Un nombre trop petit donne un résultat nul, un dépassement de capacité provoque une condition d'erreur. Exemple: 2.7E0 INT renvoie l'entier double précision 2.

F!   r adr ---

Affecte un nombre réel situé au sommet de la pile des réels à l'adresse dont la valeur est située au sommet de la pile de données.

F@   adr --- r

Prélève un nombre réel depuis l'adresse adr dont la valeur est située au sommet de la pile de données et dépose le réel sur la pile des réels.

FCONSTANT   r --- <mot>   (en compilation)
                     --- r   (en exécution)

Mot de définition utilisé sous la forme:

r FCONSTANT <mot>

Lors de l'exécution de FCONSTANT, il crée un en-tête <mot> dans le dictionnaire et affecte le nombre réel r à sa zone paramétrique. Lors de l'exécution ultérieure de <mot>, le nombre réel r qui lui a été affecté est déposé sur la pile des réels.

FVARIABLE   ---   (en compilation)
                     --- adr   (en interprétation)

VF4

Mot de définition utilisé sous la forme:

FVARIABLE ‹mot›

Lors de l'exécution de FVARIABLE, un en-tête est créé, avec une zone paramétrique explicitement non initialisée. Lors de l'exécution de ‹mot›, l'adresse du champ paramétrique est déposée sur la pile de données de manière à permettre l'action des mots F@ et F!.

Routines d'Affichage Requises en Virgule Flottante

E. r ---

L'affichage de r est réalisée en mode exposant. La mantisse contient le nombre maximum de digits admis par l'implémentation, et l'exposant est explicitement affiché s'il est différent de zéro. Un espace de séparation suit l'affichage du nombre. Si la base courante du système n'est pas décimale, il se produit une condition d'erreur. Pour exemple, avec l'implémentation de MICROMOTION, 12345.67E2 E. affichera .12334567E-07

F. r ---

L'affichage de r est réalisée en format point fixe, en général avec un ajustement de la position du point décimal si nécessaire, ceci afin de ne pas recourir à la notation en mode exposant. Le nombre de digits spécifiés par la plus récente exécution du mot PLACES sont affichés à droite du point décimal. Un espace de séparation suit l'affichage du nombre. Pour exemple, 4 PLACES 1.2345E02 F. affichera
123.4500

PLACES n ---

Indique le nombre de digits affichés par défaut à droite du point décimal lors de l'exécution de l'opérateur F. .

Commandes Optionnelles en Virgule Flottante

Les mots optionnels sont du même ordre, vis à vis du 83-Standard, que les mots de référence controlée (exemple: CLS efface l'écran qui n'est pas standard, mais appartient à la référence controlée). Ainsi, ces mots, s'ils sont implantés, doivent avoir la syntaxe et l'effet sur la pile de donnée décrits ci-après.

Opérations Transcendentales Optionnelles en Virgule Flottante

FSINH r1 --- r2

Sinus hyperbolique

FCOSH r1 --- r2

Cosinus hyperbolique.

FTANH r1 --- r2

Tangeante hyperbolique.

FASINH r1 --- r2

Sinus inverse hyperbolique.

FACOSH r1 --- r2

Cosinus inverse hyperbolique.

VF5

FATANH r1 --- r2

Tangeante hyperbolique inverse.

PI --- r

Dépose le nombre réel ¶ sur la pile, représenté avec la précision maximale en format à virgule flottante.

Operateurs Logiques Optionnels en Virgule Flottante

F0> r --- f

Vrai si le nombre en virgule flottante est supérieur à zéro. Le nombre réel est supprimé de la pile (données ou réels selon le cas), et le flag est déposé sur la pile de données.

F> r1 r2 --- f

Vrai si le réel r1 est supérieur au réel r2. Les nombres réels sont supprimés de la pile et le flag est déposé sur la pile de données.

Commandes Optionnelles d'affichage en Virgule Flottante

Suggestion d'option à l'affichage en virgule flottante: si un dépassement de capacité se produit lors de la conversion, le champ d'affichage est rempli avec des astérisques (caractère "*"). Si le nombre est trop petit, lors de la conversion, le champ d'affichage est rempli avec le caractère "U".

E.R r n1 n2 ---

Affiche r sur le périphérique courant en format exposant avec n1 digits à droite du point décimal, le tout justifié à droite dans un champ de n2 caractères. Si la base numérique courante n'est pas décimale, il se produit une condition d'erreur. Pour exemple:
1.234E0 5 12 E.R affiche
.12340E-01

F.R r n1 n2 ---

Affiche r sur le périphérique courant en format point fixe avec n1 digits à droite du point décimal, le tout justifié à droite dans un champ de n2 caractères. Les chiffres ne pouvant pas être représentés dans un tel champ apparaissent en format exposant. Si la base numérique courante n'est pas décimale, il se produit une condition d'erreur. Exemple:
1.2345E2 4 12 F.R affichera
123.4500

Operations Mixtes Optionnelles en Virgule Flottante

FNUMBER adr --- r

Adr pointe sur une chaine comptée (counted string, c'est à dire dont le premier octet indique la longueur) à fin de conversion en nombre réel. Le résultat est déposé sur la pile des réels. Si la syntaxe de la chaine est incorrecte pour un nombre en virgule flottante, il se produit une condition d'erreur et la valeur résultante est indéfinie.

PACK d n --- r

VF6

Convertit un nombre double précision en tant que mantisse et un entier 16 bits signé en tant que puissance de 10 en un nombre réel. Le

fonctionnement est dépendant de l'implémentation.

UNPACK r --- d n

Convertit un nombre réel en un nombre double précision en tant que mantisse et un nombre 16 bits en tant que puissance de 10. le fonctionnement est dépendant de l'implémentation.

F#BYTES --- n

Constante délivrant le nombre d'octets représentant un nombre réel etl qu'il est représenté sur la pile des réels. Dépend de l'implémentation.

F#PLACES --- n

Constante délivrant le nombre maximum de digits signifiants d'un nombre réel quand celui-ci est convertit en une chaîne ASCII décimale. Dépend de l'implémentation.

Opérations Optionnelles de Traitement d'Erreurs

Les indicateurs mimimaux suivants sont valides pour le programme d'application.

- overflow
- underflow
- divide by zero
- output convesrion overflow
- output conversion underflow
- attempted logarithm of zero
- attempted logarithm of number less than zero
- attempted square root of a number less than zero
- arcsin/arccos of argument outside the legal domain

FPSTAT --- adr

Adr pointe sur une zone de bits contenant les indicateurs d'erreur pour les opérations portant sur les nombres réels. Le nombre de bits de flag et leur signification est dépendante de l'implémentation.

FPERR r1 n --- r2

Est un mot différé ou vectorisé alloué à l'installation afin de permettre l'adaptation des messages d'erreurs par l'utilisateur. r1 est le résultat d'une opération entre un nombre réel et un entier n indiquant le type de l'erreur. En fonction du flag approprié dans FPSTAT, le nombre r2 est renvoyé tel que r2 est ou n'est pas égal à r1, ce qui est dépendant de l'implémentation et du type d'erreur.

Article de MARTIN TRACY
LOS ANGELES CHAPTER

Implantation des Routines en Virgule Flottante

Les blocs écrans décrits ci-après permettent l'implantation de routines permettant d'exploiter les fonctions de virgule flottante. Elles ont été développées en 83-Standard, cependant, à l'aide du contenu du bloc 2, la portabilité peut être assurée sur la majorité des systèmes FIG et 79 Standard.

VF7

```
SCR# 0
|                                                        28AUG84MJT
Float-point four-fonction single-precision math package with
four significant digits and an unlimited dynamic range.
Float-points numbers are represented by a signed mantissa and
an exponent of ten, with the exponent on top of the stack.

    Fixed-number    Stack ( ->)
        1.          1 0
        3.1415      31415 -4
    -1234500.      -12345 2

Used like 3.1415 FLOAT  12.5 FLOAT  F* F.

FLOAT assumes that a number containing a decimal point is
forced to a double-number and that the number of digits
following the decimal point is stored in the variable DPL.

SCR# 1
|                                                        28AUG84MJT

ONLY FORTH DEFINITIONS ALSO DECIMAL

|   2 LOAD  ( portability functions)
  3 5 THRU  ( input/output conversion)

ONLY FORTH DEFINITIONS

SCR# 2
|                                                        28AUG84MJT
: NIP   SWAP DROP  ;
: TUCK  SWAP OVER  ;  : S>D    DUP 0<  ;
: 2*    DUP +      ;  : D2*   2DUP D+  ;

-1  CONSTANT TRUE

: ?DO   COMPILE 2DUP COMPILE = [COMPILE] IF COMPILE 2DROP
        [COMPILE] ELSE [COMPILE] DO   ;              IMMEDIATE

: DO    COMPILE TRUE [COMPILE] IF [COMPILE] DO  ;  IMMEDIATE

: LOOP   [COMPILE]  LOOP [COMPILE] THEN ;   IMMEDIATE
: +LOOP  [COMPILE] +LOOP [COMPILE] THEN ;   IMMEDIATE
```

VF8

```
SCR#  3
| Zen Math - TRIM                              28AUG84MJT
: D10*   ( d1 -- d2)
| multiplies d1 by 10.
   D2*  2DUP D2* D2* D+  ;

: D+-  ( udn n -- dn)
| applies the sign of n to udn.
          0< IF  DNEGATE  THEN ;

: TRIM  ( dn n -- f)
| trims a double-number mantissa and an exponent of ten to
| a reasonable float number.
   >R  TUCK DABS
   BEGIN  OVER 0< OVER OR
   WHILE  0 10 UM/MOD  >R 10 UM/MOD  NIP R>  R> 1+ >R REPEAT
   ROT D+-  DROP R>  ;

SCR#  4
| Zen Math -  four functions                   28AUG84MJT
: F+  ROT 2DUP - ( expB-expA) DUP 0<
   IF  NEGATE  ROT >R NIP >R SWAP R>  ELSE  SWAP >R NIP  THEN
   >R S>D  R> DUP 0
  ?DO  >R D10* R> 1-  OVER ABS 6553 > IF LEAVE THEN  LOOP
   R> OVER + >R  IF  ROT DROP  ELSE ROT S>D D+ THEN  R> TRIM  ;

: FNEGATE  >R NEGATE R>  ;  : F-   FNEGATE F+  ;

: F*  ROT + >R
   2DUP XOR >R  ABS SWAP ABS UM*  R> D+-  R> TRIM  ;

: F/  OVER 0= ABORT" 0/"
   ROT SWAP - >R  2DUP XOR -ROT  ABS  DUP 6553 MIN  ROT ABS 0
   BEGIN  2DUP D10* DROP 3 PICK < WHILE  D10*  R> 1- >R  REPEAT
   2SWAP DROP  UM/MOD  NIP  0 ROT   D+-  R> TRIM ;

SCR#  5
| Zen Math - input and output                  28AUG84MJT
| Float numbers must include a decimal point.
| DPL contains the numbers of digits to the right of the decimal.

: FLOAT  ( n -- f)
| converts the last entered number to float format.
   DPL @ NEGATE TRIM  ;

: F.  ( f --)
| prints f in fixed format.
   >R  DUP ABS 0
   <#  R@             0 MAX 0 ?DO  ASCII 0 HOLD  LOOP
       R@ 0<
       IF   R@ NEGATE 0 MAX 0 ?DO  #  LOOP ASCII . HOLD
       THEN  R> DROP  #S  ROT SIGN
    #> TYPE SPACE ;
```

VF9

```
SCR#  6
| Zen sliderule - aliases                          16OCT84NG
|             !! IMPLEMENTATION DEPENDENT !!
| Zen math stacks a floating number in two cells with the
| signed exponent of 10 in one cell on top and the signed
| mantissa in a second cell bellow.

: D5*   2DUP D2* D2* D+  ;     : F2* 2 0 F* ;
: F2/  ( r --- r/2)  SWAP S>D D5* ROT 1- TRIM  ;
: F2EN/  ( r n --- r/2En)  0 ?DO F2/ LOOP  ;
: FDROP  2DROP  ;  : FDUP  2DUP  ;  : FOVER  2OVER  ;
: FSWAP  2SWAP  ;  : FROT  2ROT  ;  : F!  2!  ;
: F0< DROP 0<  ;   : F@   2@   ;  : F,  FLOAT  , ,  ;
: FCONSTANT  FLOAT 2CONSTANT  ;  4 CONSTANT F#BYTES
: FVARIABLE   2VARIABLE  ;
: FF  FLOAT  ;

SCR#  7
| Zen sliderule - constants and variables    16OCT84NG


0.6073 FCONSTANT 1/K
             VARIABLE  MODE_FLAG
1.2076 FCONSTANT 1/K'
             VARIABLE  DELTA_FLAG
0.0000 FCONSTANT F0

1.0000 FCONSTANT F1
             FVARIABLE F-BIN
0.2500 FCONSTANT F1/4

FVARIABLE XX FVARIABLE YY FVARIABLE ZZ

VARIABLE NDX

SCR#  8
| Zen sliderule - arrays                           16OCT84NG

: FARRAY
   CREATE  DOES>  SWAP F#BYTES * + F@  ;  | n --- nth

FARRAY  +EPS
.7854 F, .4636 F, .2450 F, .1244 F, .06242 F, .03124 F,
.01562 F, .007812 F, .003906 F, .001953 F, .0009766 F,
.0004883 F, .0002441 F, .0001221 F, .00006104 F,

FARRAY -EPS
.0000 F, .5493 F, .2254 F, .1257 F, .06258 F, .03126 F,
.01563 F, .007813 F, .003906 F, .001953 F, .0009766 F,
.0004883 F, .0002441 F, .0001221 F, .00006104 F,

DEFER EPS  | it will be either +EPS or -EPS
```

VF10

```
SCR#  9
| Zen sliderule - stack manipulators          16OCT84NG

: DELTA_SIGN  DELTA_FLAG @  IF FNEGATE THEN  ;

: MODE_SIGN  MODE_FLAG  @ IF FNEGATE THEN  ;

: R-DELTA=  ( r --- r)  FDUP  F0<  DELTA_FLAG  !  ;

: V-DELTA=  ( r --- r)  FDUP F0< NOT DELTA_FLAG !  ;

: R-STORE  ( x y z --- ) R-DELTA= ZZ F! YY F!  ;

: V-STORE  ( x y z --- )  ZZ F!  V-DELTA=  YY F!  XX F!  ;

DEFER !STACK  | it will be R-STORE or V-STORE


SCR#  10
| Zen sliderule - stack manipulators          16OCT84NG

: NEW_Z   ZZ F@  NDX @ EPS DELTA_SIGN  F-   ;

: NEW_X  XX F@ YY F@ NDX @ F2EN/  DELTA_SIGN MODE_SIGN F+  ;

: NEW_Y  YY F@ XX F@ NDX @ F2EN/  DELTA_SIGN     F+  ;

: ROT'ING  NEW_X NEW_Y NEW_Z  R-STORE  ;

: VEC'ING  NEW_X NEW_Y NEW_Z  V-STORE  ;


DEFER DO-IT  | it will be either ROT'ING or VEC'ING


SCR# 11
| Zen sliderule - cordic algorithm            16OCT84NG

: MODE_=_+1
  -1 MODE_FLAG !  (') +EPS IS EPS  ;

: MODE_=_-1
  0 MODE_FLAG ! (') -EPS IS EPS ;

: CORDIC | xstrart ystart zstart --- xend yend zend
   !STACK MODE_FLAG @ DUP 0= >R
          IF   0 NDX !  DO-IT  THEN
     4 1  DO   I NDX !  DO-IT  LOOP
      R@  IF   4 NDX !  DO-IT  THEN
    14 4  DO   I NDX !  DO-IT  LOOP
      R>  IF  13 NDX !  DO-IT  THEN
              14 NDX !  DO-IT  XX F@ YY F@ ZZ F@  ;
```

VF11

```
SCR#  12
| zen sliderule - cos, sin, tan, cosh, sinh, tanh, exp 16OCT84NG
: FCOS&SIN  | r --- cos r sin r
   1/K FSWAP F0 FSWAP  (') R-STORE IS !STACK
   MODE_=_+1  (') ROT'ING IS DO-IT  CORDIC  ;
: FCOS  ( r --- cos r)  FCOS&SIN        FDROP FDROP ;
: FSIN  ( r --- sin r)  FCOS&SIN FDROP FSWAP FDROP ;
: FTAN  ( r --- tan r)  FCOS&SIN FDROP FSWAP F/    ;

: FCOSH&SINH  | r --- cosh r sinh r
   1/K' FSWAP F0 FSWAP  (') R-STORE IS !STACK
   MODE_=_-1  (') ROT-ING IS DO-IT  CORDIC  ;
: FCOSH  ( r --- cosh r)  FCOSH&SINH        FDROP FDROP ;
: FSINH  ( r --- sinh r)  FCOSH&SINH FDROP FSWAP FDROP ;
: FTANH  ( r --- tanh r)  FCOSH&SINH FDROP FSWAP F/ ;
: FALN   ( r --- exp r)  FCOSH&SINH         FDROP F+ ;

SCR#  13
| Zen sliderule - sqrt, ln, atanh              16OCT84NG

: FLN   | r --- ln r
   FDUP F1  F+ FSWAP F1 F- F0 (') V-STORE IS !STACK
   MODE_=_-1  (') VEC'ING IS DO-IT  CORDIC
   FSWAP FDROP FSWAP FDROP F2*  ;

: FSQRT  | r --- sqrt r
   FDUP F1/4 F+ FSWAP F1/4 F- F0 (') V-STORE IS !STACK
   MODE_=_-1  (') VEC'ING ID DO-IT  CORDIC
   FDROP FDROP 1/K' F* ;

: FATANH   | r --- argtanh r
   (') V-STORE IS !STACK
   F1 FSWAP F0  MODE_=_-1  (') VEC'ING IS DO-IT  CORDIC
   FSWAP FDROP FSWAP FDROP ;

SCR#  14
| Zen sliderule - p>r, r>p, atan               16OCT84NG

: R>R  | x y --- (xE2 + yE2)E1/2 arctan (y/x)
   1/K F* FSWAP 1/K F* FSWAP F0  (') V-STORE IS !STACK
   MODE_=_+1  (') VEC'ING IS DO-IT  CORDIC FSWAP FDROP ;

: FATAN  ( r --- arctan r)  F1 FSWAP R>P FSWAP FDROP ;

: P>R  | radius angle --- x y
   FOVER FSWAP FCOS&SIN FDROP
   FROT F*  FROT FROT F*  FSWAP  ;
```

Le contenu des écrans 1 à 5 est dû à MARTIN TRACY du chapitre FORTH de LOS ANGELES.

Le contenu des écrans 6 à 14 est dû à NATHANIEL GROSSMAN, Departement of Mathematics, UCLA, LOS ANGELES, CA 90024

Nous espérons vous expliquer en détail le fonctionnement de l'algorithme de CORDIC à partir duquel ont été obtenues les fonctions sin, cos, tan, sinh, cosh, tanh, exp, ln et sqrt.

S'il y a des volontaires pour assurer la traduction de l'article (écrit en anglais) actuellement en notre possession, ils seront les bienvenus.

VF12

X NOTION DE PROGRAMME (ou ROUTINE) EN MUMPS

Nous venons d'étudier la constitution des lignes de commandes. Nous allons passer maintenant à un niveau supérieur d'organisation, pour le code de MUMPS, que nous nommerons routine. En fait, une routine n'est, ni plus ni moins, qu'une suite de lignes de commandes associées à un nom unique (nom de programme). Ce nom répond à la même règle de définition que celle déjà vue pour les variables.

Le stockage, le chargement ou l'accès à ces routines seront étudiés ultérieurement. Par convention, en MUMPS, la première ligne d'une routine contiendra une étiquette équivalente à son nom, un commentaire incluant le nom de l'auteur, la date de création et la dernière date de mise à jour. Pour des raisons de documentation, le commentaire inclura une description sommaire de sa fonctionnalité. Toutes les lignes de commmandes seront exécutées dans la sequence de leur apparition.

De temps en temps, il est nécessaire de pouvoir modifier la sequence d'exécution des lignes ou des commandes. A cet effet, nous allons étudier deux commandes de déroutement de séquence. Ce sont les commandes DO et GOTO.

A) La commande GOTO

Parfois, il est necessaire d'ignorer un block de commandes pour traiter un cas particulier. La commande GOTO peut être employée de deux manières. Elle peut correspondre à un ordre inconditionnel ou, si elle est employée en conjonction avec l'ordre IF ou une postcondition, elle représente un ordre de branchement conditionnel. On peut imager ces deux possibilités en disant :

- allez fermer la porte !

- si vous avez froid, allez fermer la porte !

En MUMPS, le branchement se fera à une étiquette ou à une ligne spécifiée dont la position est relative à une etiquette. Le petit programme suivant permet d'expliquer comment on peut utiliser la commande GOTO. Comme toutes les commandes, GOTO peut être mentionnée avec sa premiere lettre; à savoir G.

```
DEBUT     ;exemple de la commande goto
          R !,"veuillez entrer  votre sexe (M/F) :",SEXE I SEXE["M" G MALE
          G:"F"'[SEXE DEBUT+1
          R !,"avez-vous eu des enfants (O/N) ",REP1
          R !,"avez-vous vos regles regulierement (O/N) ",REP2 G SUITE
MALE      R !,"etes-vous circoncis (O/N) ",REP3
          R !,"avez-vous eu des problemes de prostate (O/N) ",REP4
SUITE     R !,"est ce que vous fumez (O/N) ",REP5
          I REP5="O" R "Rarement, Moderement, Beaucoup (R/M/B) ",REP6
```

Note : Nous avons volontairement utilisé la forme simplifiée des commandes puisque c'est la forme couramment employée, en MUMPS, par les programmeurs.

Dans cet exemple, il est facile de voir l'utilité de la commande GOTO pour orienter les séquences de questions. Incidemment, l'une des utilisations générales du langage MUMPS est de pouvoir, grâce à une série de questions, obtenir un renseignement complet (ex. un historique de patient). Les puristes vont s'empresser de dire que MUMPS n'est pas un langage structuré puisqu'il supporte l'ordre GOTO . Les prochaines instructions permetteront de dire le contraire.

B) Les commandes DO et QUIT

La commande DO permet également de transférer l'exécution à une ligne specifiée. A la différence de l'ordre GOTO, il est possible de continuer en séquence les instructions citées après l'ordre de débranchement. Le retour est effectué à la rencontre de l'ordre QUIT. Si plusieurs ordres DO sont imbriqués, le premier ordre QUIT correspond au dernier ordre DO. Un exemple est donné dans la petite routine suivante :

→

```
ROUT1     ;analyse d'une journee Y.L.G. Ecrit:1/oct/84  Maj:2/nov/84 10h00
          W #,!,"nous voulons des info. Sur votre vie de tous les jours"
          W !,"entrez le nombre d'heures passees aux occupations suivantes"
          W !,"pour les minutes entrez le temps en centieme"
          W !,"combien d'heures passez-vous a : "
DEBUT     R !,"dormir : ",V D CALCUL S DORMIR=C
          R !,"lire ou etudier : ",V D CALCUL S LIRE=C
          R !,"vous deplacer : ",V D CALCUL S TRANS=C
          R !,"manger et vous detendre : ",V D CALCUL S MANGE=C
          R !,"faire du sport : ",V D CALCUL S SPORT=C
          S TOTAL=+DORMIR+LIRE+TRANS+MANGE+SPORT
          I TOTAL>.9 G OK
          E  R !,"que faites vous le reste du temps ? ",AUTRE
OK        I TOTAL>1 W !,"une journee n'a que 24 heures,recommencez" G DEBUT
          W !,"vous etes bien occupe, merci de ces reponses" G FIN
CALCUL    S C=V/24
FIN       ;fin de la routine
```

Nous allons faire quelques remarques au sujet de cette routine. Si on l'analyse de près, on s'aperçoit qu'une seule question sera posée. Après l'exécution de la ligne calcul, c'est la ligne fin qui est exécutée. Comme il n'y a plus de ligne après la ligne fin, le traitement s'arrête. Pour accomplir le traitement de la manière prévue, il suffit d'écrire la ligne CALCUL de la façon suivante :

```
CALCUL    S C=V/24 Q
```

Nous avons dit précédemment que MUMPS etait un langage permettant la programmation structurée. En effet, en utilisant les postconditions il est très simple de simuler les instructions DO WHILE ou DO UNTIL. Prenons l'exemple suivant :

```
DEBUT     S SOIF="O"
TANTQUE   Q:SOIF'="O"  R !,"avez-vous soif ?,SOIF D:SOIF="O" BOIRE G TANTQUE
FIN       W !,"vous etes deja saoul ..." Q
BOIRE     W !,"alors, buvez" Q
```

L'exemple ci-dessus illustre le phenomène : tant que vous avez soif, buvez ! Notez l'utilisation de QUIT. L'ordre QUIT indique la fin du niveau dans lequel on est. Si on le trouve, suite a un ordre DO l'exécution est transférée à la commande située après le DO. Dans les autres cas l'exécution de la ligne est arrêtée. Pour être plus clair, décomposons ce programme action par action et ligne par ligne :

- affectation de la chaine "O" à la variable SOIF

- abandon de la ligne si SOIF ne contient pas "O". Envoi de la question avez-vous soif ? . Attente d'une saisie clavier. Celle-ci est affectée à la variable SOIF. Debranchement à ce qu'on appele un sous-programme, si SOIF est égale a "O". Ce sous-programme affiche le message " alors,buvez" ;puis revient pour l'exécution de l'ordre suivant. Soit, l'ordre de branchement à la meme ligne.

- envoi du message " vous êtes déjà saoûl" et abandon de l'exécution du programme.

- la ligne BOIRE est à considérer comme un sous-programme.

Note: l'intruction QUIT n'a jammais d'argument. Elle sera séparée des verbes suivants, s'ils existent, par deux espaces.

C) Postcondition(s) dans les arguments

Nous avons déja vu qu'il etait possible de postconditionner les verbes. Il est également possible, pour certains verbes de mettre des postcoditions dans les arguments (DO, GOTO, XECUTE). En plus, on peut avoir les deux types de postconditions. A savoir, postconditionner le verbe et postconditionner ses arguments. Exemple :

```
TEST      R !,"données ok (O/N),X
          G:X'="" OK:X="O",FAUX:X="N",TEST W !," ?  " G TEST
          Q
OK        W !,"les données sont bonnes" G FIN
FAUX      W !,"les données sont fausses"
FIN       ;fin de la routine
```

Dans cet exemple, nous avons utilisé les deux possibilités de postconditionnement.

D) Les commandes HALT et HANG

Ces deux commandes qui s'écrivent dans leurs formes abrégées de la même manière (H), exécutent des actions totalement différentes. Il est nécessaire, de temps en temps, de pouvoir arrèter un traitement en cours, lorsque l'ordinateur, parce que mal programmé, devient une machine folle. C'est le rôle de la commande HALT. Celle-ci sera citée sans argument, si d'autres verbes la suivent, HALT sera séparée du verbe suivant par deux espaces. Quand MUMPS recontre cette commande, il arrète toutes les exécutions en cours et retourne sous le contrôle du système. HALT, comme toutes les commandes, peut etre poscondítionnée. La commande HANG est moins sauvage. Elle permet de suspendre un traitement, non pas definitivement, mais pendant un certain nombre de seconde(s) specifiée(s). MUMPS temporisera silencieusement puis reprendra, en sequence, le traitement. Pour illustrer l'utilisation de cette commande, imaginez qu'un grand texte soit envoyé en continu sur l'écran. Il est intéressant de pouvoir temporiser en fin d'écran pour vous laisser le temps de lire ce texte.

RESUME

Nous avons maintenant tous les éléments nécessaires pour écrire une routine MUMPS. Il est évident que nous n'avons pas vu toutes les composantes de MUMPS. Dans ce chapitre nous avons vu :

Qu'une routine est un ensemble de lignes de commandes. Qu'elle est identifiée par un nom unique, que sa première ligne contient son nom, son auteur, sa date de création ainsi que sa date de mise à jour. Les lignes de commande peuvent être labellées ou non. Les debranchements (DO GOTO) peuvent se faire à une position relative à une étiquette. Dans les prochains chapitres nous verrons comment accéder aux routines, ce qu'est un fichier en MUMPS et d'autres éléments permettant de se construire des outils.

L'EXECUTION VECTORISEE

Sous cette appellation un peu ésotérique se cache un concept tout à fait nouveau et un peu révolutionnaire pour celui qui ne pratique pas les nouveaux langages. Mais pour préciser un peu notre propos, revenons au nidu langage machine et décrivons dans le détail un mécanisme un peu troublant pour le programmeur débutant: l'exécution par indirection.

Le microprocesseur choisi est le 6809 pour la richesse de ses modes d'adressages.

```
          ORG       EMDMEM-$400
PUTC      EQU       $E803

START     LDB       ENDMEM-$200
          JSR       PUTC
          END
```

Dans ce petit exemple, le registre B est rempli avec la valeur de l'adresse mémoire ENDMEM-$200 et l'exécution passe au sous-programme PUTC qui est chargé d'afficher le caractère dont le code est dans le registre B (ceci sur TO7 ou TO7/70)

Une fois la routine implantée en mémoire, elle peut être utilisée par un langage évolué:

en BASIC: CALL (ENDMEM-&H400)

en FORTH: ENDMEM @ HEX 400 - CALL

Mais si on désire que notre routine "filtre" les caractères affichés de manière à ne prendre en compte que les caractères ASCII affichables et transforme les codes de contrôle en espaces, il faut réécrire notre routine.

```
          ORG       ENDMEM-$400
PUTC      EQU       $E803

START     LDB       ENDMEM-$200
          CMPB      #32
          BLS       AFFECTE
          BRA       AFFICHE
AFFECTE   LDB       #32
AFFICHE   JSR       PUTC
          END
```

Cette petite gymnastique nous oblige, à partir de BASIC ou FORTH, à tenir compte de deux adresses si on veut ou non filter notre affichage, à moins de faire appel à une indirection:

```
          ORG       ENDMEM-$400
PUTC      EQU       $E803
VECTEUR   EQU       ENDMEM-$100
START     LDB       ENDMEM-$200
          JMP       [VECTEUR]
SUITE     CMPB      #32
          BLS       AFFECTE   (si B<ou= à 32)
          BRA       AFFICHE
AFFECTE   LDB       #32
AFFICHE   JSR       PUTC
          END
```

Dans ce cas, l'exécution se poursuit à l'adresse pointée par le contenu de VECTEUR. Si cette adresse a été initialisée comme suit:

en BASIC: DOKE VECTEUR,SUITE
ou DOKE VECTEUR,AFFICHE

(DOKE=POKE adr, POKE adr+1; n'existe pas sur sur THOMSON; tant pis pour vous ...)

en FORTH: SUITE VECTEUR !
ou AFFICHE VECTEUR !

Ainsi, un START CALL exécute un filtrage ou non ceci en fonction du contenu de VECTEUR. Comment ce peut-ce ???

Cette première approche avait pour seul but de vous faire sentir certaines subtilités. Ne vous fatiguez pas à expérimenter ce qui précède, la suite est bien plus intéressante et s'applique à tous les systèmes.

MONSIEUR LE PORTIER D'HOTEL, BONJOUR...

L'exécution sans indirection correspond à la situation suivante:

- j'arrive à l'hotel et je demande la chambre qui m'est réservée. Le portier me dit que c'est celle qui porte le numéro 32.

L'exécution avec indirection s'apparente à une situation nettement moins courante:

j'arrive dans le même hôtel, mais le portier qui est un original, me donne pour seule indication, le numéro de la chambre 10, en me précisant que dans cette chambre je trouverai un papier indiquant le numéro de ma chambre.

Du DEVOS tout ceci !!!

Ne nous trompons pas, JMP VECTEUR et JMP [VECTEUR] n'ont pas le même comportement.

Ce mécanisme n'est pas reproductible en BASIC vers un sous-programme BASIC, à l'exception d'une seule machine, le ZX81 qui accepte les GOTO et GOSUB suivis d'une étiquette:

```
10 LET VECTEUR=5000
20 GOSUB VECTEUR
        etc....
```

En FORTH, par contre, il existe des commandes permettant l'exécution vectorisée de routines à travers une définition accessible à l'utilisateur.

DEFINITION DE VECTEURS EN 83-STANDARD

La définition d'un vecteur, en 83-STANDARD est réalisée par le mot DEFER :

```
DEFER <     --- <mot>
```

Crée un en-tête et réserve deux octets dans la zone paramètre du mot. L'exécution du <mot> exécute le mot dont le cfa se trouve dans la zone paramètre de <mot>.

L'affectation peut être réalisée par le mot ['] et IS.

Exemple:

```
: NOUVEAU-EMIT  ( n ---)
DUP 32 < IF DROP 32 THEN EMIT ;

DEFER C-EMIT

['] NOUVEAU-EMIT IS C-EMIT
```

à l'exécution:

```
  65 C-EMIT affiche le caractère "A"
  13 C-EMIT affiche un espace
```

Une réaffectation de C-EMIT:

```
['] EMIT IS C-EMIT
```

à l'exécution:

```
  65 C-EMIT affiche le caractère "A"
  13 EMIT provoque un retour chariot
```

Ainsi, C-EMIT exécute NOUVEAU-EMIT ou EMIT selon le cfa qui lui a été affecté.

Pour ceux qui programment sur des systèmes de type 79-Standard ou FIG, les mots DEFER, IS et ['] peuvent se définir de la manière suivante:

```
: DEFER  ( --- <mot>)
CREATE  ( ou <BUILDS en FIG)
  0 ,
DOES>
  @   ?DUP
   IF    EXECUTE
   ELSE  ." NON INITIALISE" CR  THEN ;

: [']
[COMPILE] ' 2 -
STATE @
IF
    [COMPILE] LITERAL THEN  ;    IMMEDIATE
```

```
: IS
[COMPILE] '
STATE @
IF        [COMPILE] LITERAL
          COMPILE !
ELSE      !    THEN ;
IMMEDIATE
```

DOMAINES D'APPLICATION

La vectorisation permet d'exécuter "en différé" une définition non encore compilée (d'où le mot DEFER, pour DIFFERER -en français-), mais aussi de faire exécuter de diverses manières un même mot sans avoir à modifier les définitions déjà compilées.

Voyons ceci avec un exemple concret. De nombreux systèmes FORTH délivrent les codes d'erreurs uniquement sous forme de nombres, ce qui ne facilite guère la vie du programmeur que vous êtes et qui aime les choses claires. En FORTH, le mot qui délivre les codes ou messages s'appelle MESSAGE. Exemple 1 MESSAGE affiche 1 ou EMPTY-STACK. Mais certains messages ne sont pas toujours explicites même en anglais. Si votre FORTH est écrit en mémoire vive, vous pouvez redéfinir MESSAGE comme suit:

```
DEFER (MESSAGE)

['] (MESSAGE) ' MESSAGE !
['] ;S ' MESSAGE 2 + !

: MESSAGE-EN-FRANCAIS   ( n --- )
CASE
  1 OF     ." PILE VIDE"              ENDOF
  2 OF     ."        etc.....
.........
ENDCASE  CR  ;

 '  MESSAGE-EN-FRANCAIS IS (MESSAGE)
```

et maintenant, 1 MESSAGE affiche PILE VIDE.

Et si çà vous chante, vous pouvez définir des messages d'erreurs propres à vos applications ou dans une langue étrangère. Ainsi, si vous avez également défini des messages en allemand et en anglais (et plus encore), vous pouvez définir:

```
: CHOIX-LANGAGE ( -  - )
CR ." CHOISISSEZ - CHOOSES - WÄHLEN:"
CR CR
    ." 1...FRANCAIS - FRENCH - FRANZÖSICH"
CR CR ." 2...ANGLAIS - ENGLISH - ENGLISH "
CR CR ." 3...ALLEMAND  GERMAN - DEUTSCH"
KEY  CASE
" 1" ASC OF ['] F-MSG IS (MESSAGE) ENDOF
" 2" ASC OF ['] G-MSG IS (MESSAGE) ENDOF
" 3" ASC OF ['] D-MSG IS (MESSAGE) ENDOF
MESSAGES-NUMERIQUES
ENDCASE  ;
```

Dans l'exemple ci-dessus, F-MSG correspond aux messages en français, G-MSG ceux en anglais et D-MSG ceux en allemand. MESSAGE-NUMERIQUES sélectionne l'option messages chiffrés par défaut si on choisit une option différente de celle affichée.

L'intérêt est encore plus évident pour certains cas critiques, tels l'usage d'imprimantes dont tout le monde à eu affaire aux codes de contrôles capricieux au possible. Voici le cas le plus fréquent: la séquence PRINTER 13 EMIT CONSOLE provoque selon l'imprimante et les options sélectionnées sur celle-ci, un retour chariot simple ou un retour chariot avec descente à la ligne suivante (CR+LF, soit 13 EMIT 10 EMIT).

Afin de remédier à ce dilemne, nous allons redéfinir CR à partir d'une primitive (CR) ou (CR+LF) de manière vectorisée:

```
: (CR) CR ;
: (CR+LF) CR 10 EMIT ;

DEFER CR

: MANESSMAN-SELECT
['] (CR) IS CR  ;
: SEIKOSHA-SELECT
['] (CR+LF) IS CR  ;
 : PRINTER-SELECT  ( ---)
CR ." FAITES VOTRE CHOIX:" CR CR
."  1...Sélection de l'imprimante MANESSMAN" CR
."  2...Sélection de l'imprimante SEIKOSHA" CR
KEY CASE
  1 OF MANESSMAN-SELECT ENDOF
  2 OF SEIKOSHA-SELECT  ENDOF
ENDCASE ;
```

Naturellement, ceci reste à adapter aux particularités de votre système. Ainsi, pour ne pas rendre votre affichage perturbant lors du retour à l'affichage vidéo, il faut modifier les routines ci-dessus:

```
: (CR) 13 EMIT ;
: (CR+LF) 13 EMIT 10 EMIT ;

DEFER CR
DEFER SELECTION

: MANESSMAN-SELECT   ['] CR IS (CR) ;
: SEIKOSHA-SELECT    ['] CR IS (CR+LF) ;

: PRINTER-SELECT  ( ---)
CR ." FAITES VOTRE CHOIX:" CR CR
." 1...Sélection de l'imprimante MANESSMAN" CR
." 2...Sélection de l'imprimante SEIKOSHA" CR
KEY CASE
  1 OF ['] SELECTION IS MANESSMAN-SELECT ENDOF
  2 OF ['] SELECTION IS SEIKOSHA-SELECT  ENDOF
ENDCASE ;

: (PRINTER) ( reprise ancienne version )
PRINTER     ( en tant que primitive ) ;

: PRINTER SELECTION (PRINTER) ;

: (CONSOLE) ( reprise ancienne version )
CONSOLE     ( en tant que primitive ) ;

: CONSOLE ['] CR IS (CR+LF)  (CONSOLE) ;
```

Bien entendu, tout ceci est une illustration de ce qui se passe sur un système hébergeant le langage FORTH en mémoire vive tels les systèmes APPLE, ORIC, AMSTRAD et tous systèmes sous CP/M équipés du F83 (Newbrain, KAYPRO...etc...).

Mais qu'en est il pour les systèmes hébergeant les primitives dans une cartouche ou en ROM ? (cas des THOMSON, HECTOR HRX, certaines versions pour COMMODORE, SINCLAIR QL, etc....).

La solution dépend des orientations prises par le constructeur. Personnellement, je ne puis citer que le FORTH THOMSON, car je le maitrise à peu près.

LES TABLEAUX DE VECTEURS

Le langage FORTH des systèmes THOMSON se présentant sous forme de cartouche, il n'est pas possible de modifier les adresses d'exécution. Mais le manuel de référence apporte des informations précieuses à celui qui le parcourt, nottament au chapitre ANNEXE D, page 231. Certains mots sont dits "mots appelants" et exécutent un "mot appelé par défaut".

Les codes d'exécution des mots appelés par défaut sont implantés dans un tableau de vecteurs appelé SYSVEC. En fait, SYSVEC est plutôt un mot qui calcule un décalage par rapport à l'origine du tableau. Ainsi, 0 SYSVEC @ délivre le cfa (code d'exécution) du mot (CR). Donc, les séquences 0 SYSVEC @ EXECUTE, ou (CR) ou CR sont équivalentes. En fait, la définition de CR une fois décompilée présente la définition:

```
: CR 0 SYSVEC @ EXECUTE ;
```

ce qui équivaut presque à:

```
: CR (CR) ;
```

Mais ce qui est subtil dans cette technique est la possibilité offerte à l'utilisateur de définir des mots en langage évolué qui se substitueront aux primitives.

Exemple:

1 SYSVEC @ délivre le cfa de (EMIT) qui est le mot appelé par défaut par EMIT. Voici une plaisanterie bien innocente à réaliser: →

```
 : (NOUVEAU EMIT) DUP (EMIT) (EMIT) ;
FIND (NOUVEAU EMIT) 1 SYSVEC !
```

et à partir de maintenant, votre THOMSON est pris de baigaiement. Si vous essayez de taper VLIST, vous obtiendrez VVLLIISSTT et le résultat sera du même ordre (OOKK).

rétablissez la situation en tapant:

```
 FIND (EMIT) 1 SYSVEC !
```

que vous verrez provisoirement sous la forme:

```
 FFIINNDD ((EEMMIITT))   etc...
```

Voici un exemple illustrant à merveille les possibilités de cette technique. Le problème à résoudre est le suivant: il s'agit d'imprimer sur une SEIKOSHA SP800 (compatible SP1000 et l'imprimante livrée avec le TO9 !!) connectée à un TO7 ou TO7/70, toutes les accentuées apparaissant normalement à l'écran.

le mot (PEMIT-F) transcode les caractères compris entre 129 et 151 en leur équivalent accentués SEIKOSHA (codes IBM). Sur le TO7, en FORTH "é" a pour code 134 et sur la SP800, le code 130. Deux options complémentaires ont été sélectionnées:

- le code 127 sur le TO7 transmet la séquence de passage en caractères gras sur la SP800.
- le code 166 réalise l'inverse du code 127.

Afin de court-circuiter la fonction (PEMIT) qui assure un filtrage des accentués du moniteur en leur équivalents non accentués vers l'imprimante, on appelle directement la routine PUTC:

```
 REGD ! [ HEX ] E812 CALL [ DECIMAL ]
```

Les mots ACC-ON et ACC-OFF valident et dévalident la possibilité d'obtention de caractères accentués sur l'imprimante.

Le mot (NKEY) permet d'accéder aux caractères 127 et 166 directement à partir du clavier. En prime, tout appui sur la touche marquée [↰] suivi d'un caractère affiche le caractère correspondant à celui de la touche activée, mais augmenté de 128. Si vous avez réalisé une redéfinition graphique faisant correspondre les caractères affichés à ceux de l'imprimante, vous aurrez la possibilité d'imprimer et composer des textes contenant des caractères graphiques et mathématiques:

$$\int dx\left(\frac{1}{x\hat{}3}\right)$$

≤Σ μτ⊊Ωδ∞Ø¼Γεπ±β

```
SCR: 9
: (PEMIT-F)  ( n --- n' ) CASE
  " ü" ASC OF  129 ENDOF
  " é" ASC OF  130 ENDOF
  " à" ASC OF  133 ENDOF
  " â" ASC OF  134 ENDOF
  " ç" ASC OF  135 ENDOF
  " ê" ASC OF  136 ENDOF
  " ë" ASC OF  137 ENDOF
  " è" ASC OF  138 ENDOF
  " ï" ASC OF  139 ENDOF
  " î" ASC OF  140 ENDOF
  " ô" ASC OF  147 ENDOF
  " ö" ASC OF  148 ENDOF
  " û" ASC OF  150 ENDOF
  " ù" ASC OF  151 ENDOF
  127 OF 27 (PEMIT) 69 (PEMIT) 32 ENDOF
  166 OF 27 (PEMIT) 70 (PEMIT) 32 ENDOF
  DUP ENDCASE
REGD ! [ HEX ] E812 CALL [ DECIMAL ] ;

: ACC-ON    ( --- )
[ FIND (PEMIT-F) ] LITERAL 9 SYSVEC ! ;

: ACC-OFF   ( --- )
[ FIND (PEMIT) ]   LITERAL 9 SYSVEC ! ;


SCR: 50
: (NKEY)    ( --- )
(KEY) DUP
CASE
  30 OF DROP (KEY)
          128 +  ENDOF ( Action 30    )
  12 OF DROP 127 ENDOF ( Action CTR-L)
  18 OF DROP 166 ENDOF ( Action CTR-R)
ENDCASE  ;


FIND (NKEY) 2 SYSVEC !

( La valeur 127, générée par l'action  )
( CTRL-L affiche un carré noir.        )
( La valeur 166, générée par l'action  )
( CTRL-R affiche un carré blanc.       )
```

ERREURS:

Jedi £ 19, page 2 : lire 6 45 REDUIT . . 2 15
Plus amusant : pgcd est une application directe de la récursivité qui en Forth-83 peut s'écrire :

```
: PGCD ( n1 n2 -- n )   SWAP OVER MOD  ?DUP IF MYSELF THEN ;

( pour info  : MYSELF  LATEST NAME) , ;  IMMEDIATE )
```

Bientôt un dessin animé sur votre écran ?

1/ Techniques d'animation

Les ordinateurs personnels sont des petites merveilles lorsqu'ils se mettent à animer les écrans couleur, et on trouve dans le commerce toutes sortes de jeux qui mettent à profit leurs possibilités d'animation graphique et colorée.

L'utilisateur qui, lassé des logiciels tout faits, décide de créer lui-même ses propres animations, est tout heureux de trouver dans la panoplie du "langage de programmation livré avec l'appareil" des primitives telles que PEN, PAPER, SCREEN, MODE, PLOT, DRAW, BOX, CIRCLE et autres SPRITE.

Il est vrai qu'un programme de quelques lignes est souvent déjà capable de faire apparaître des figures fort intéressantes.

Là où les choses se gâtent, c'est lorsque vous essayez d'animer votre figure en cherchant à en déplacer certains éléments. Bien sûr, vous pourriez penser prendre tout votre temps, en imitant Walt Disney, pour élaborer les plans successifs, dont l'enchaînement rapide donnerait ensuite l'illusion du mouvement.

Mais ce n'est pas ainsi que votre ordinateur procède, car chaque image prise individuellement occupe une place considérable (8 Koctets sur un Commodore 64, 32 Koctets sur un QL Sinclair) et à raison de 25 images par secondes, votre dessin animé ne pourra pas durer longtemps faute de place disponible en mémoire vive. Il existe davantage d'espace sur la mémoire de masse (cassette ou disquette), mais les temps de chargements deviennent la contrainte (7 secondes pour charger une image sur micro-cassette du QL). La solution idéale serait d'utiliser un magnétoscope ou mieux encore un disque optique numérique, mais il faut pour cela attendre la baisse des prix...

Avec les technologies d'aujourd'hui, il faut trouver des moyens de comprimer l'image, par exemple en distinguant d'une part un ou quelques décors, et d'autre part des figures animées dont on décrira le mouvement par des algorithmes. Mais ici se cache une nouvelle difficulté : une vitesse d'exécution insuffisante, qui vous contraint à passer au langage-machine.

Pour ceux qui cherchent un compromis entre les langages évolués, trop lents, et les assembleurs, trop rudimentaires, nous proposons ici une solution: le langage LPB (Langage Pseudo-Basic), qui permet, après avoir mis au point les algorithmes de mouvement en langage évolué (le BASIC), de les transposer le plus directement possible dans le langage spécifique du micro-processeur, seul moyen de gagner simultanément de la place en mémoire et de la vitesse d'exécution. La fabrication du code binaire exécutable est confiée à un compilateur, appelé BALCOM (Basic-like Assembly Language Compiler) en principe infaillible, qui vous apporte en plus d'autres avantages: vous pouvez glisser un maximum de commentaires et d'explications dans votre programme-source écrit en LPB, que vous conserverez par devers vous, tandis que vous ne livrerez à la curiosité des utilisateurs qu'un paquet de binaire exécutable particulièrement compact et très difficile à décrypter.

Voyons maintenant un exemple.

2/ Vous avez dit "scroll" ?

Pour donner l'illusion d'un mouvement rapide de caméra, il faut pouvoir faire bouger rapidement l'image dans toutes les directions. Imaginez-vous dans la cabine de pilotage d'un quelconque engin. Vous actionnez un quelconque manche à balai et le paysage qui est devant vous se met à défiler vers le haut, vers le bas, vers la gauche ou vers la droite selon le cas.

En informatique, on appelle cela un "scroll". Au cinema, quand le mouvement est horizontal, on parle plutôt de "panoramique". →

Dans l'exemple qui suit, le manche à balai sera figuré par les quatre flèches directionnelles de votre ordinateur, que vous pourrez facilement remplacer par une manette de jeu, pourvu que vous connaissiez les codes-caractère correspondants.

3/ Solution BASIC (pour un ordinateur MSX)

L'écran d'un ordinateur MSX, en mode texte, comporte 25 lignes de 40 colonnes chacune, soit 1000 cases à faire bouger, en y recopiant le contenu de la case voisine. L'algorithme correspondant est facile à mettre au point en BASIC, puisqu'il suffit de taper "RUN" pour vérifier que l'on effectue les 1000 copies dans le bon ordre.

Les primitives BASIC à utiliser sont

VPEEK(X*40+Y) qui fournit le contenu de la case écran de coordonnées X (colonne) et Y (ligne)

VPOKE X*40+Y,A qui écrit le caractère A dans la case de coordonnées X et Y

```
1 GOTO 10
2 SAVE"scroll.bas",A:END
10 A$=INKEY$:IF A$="" THEN 10
20 ON ASC(A$)-27 GOSUB 1100,1200,1300,1400
30 GOTO 10
1100 PRINT"vers la droite"
1110 FOR X=38 TO 1 STEP -1
1120 FOR Y=0 TO 23
1130  J=Y*40+X
1140  VPOKE J+1,VPEEK(J)
1150 NEXT:NEXT
1160 FOR Y=0 TO 23
1170  VPOKE 1+Y*40,0
1180 NEXT
1190 RETURN
1200 PRINT"vers la gauche"
1210 FOR X=2 TO 39
1220 FOR Y=0 TO 23
1230  J=Y*40+X
1240  VPOKE J-1,VPEEK(J)
1250 NEXT:NEXT
1260 FOR Y=0 TO 23
1270  VPOKE 39+Y*40,0
1280 NEXT
1290 RETURN
1300 PRINT"vers le haut"
1310 FOR Y=1 TO 23
1320 FOR X=1 TO 39
1330  J=Y*40+X
1340  VPOKE J-40,VPEEK(J)
1350 NEXT:NEXT
1360 FOR X=1 TO 39
1370  VPOKE 920+X,0
1380 NEXT
1390 RETURN
1400 PRINT"vers le bas "
1410 FOR Y=22 TO 0 STEP -1
1420 FOR X=1 TO 39
1430  J=Y*40+X
1440  VPOKE J+40,VPEEK(J)
1450 NEXT:NEXT
1460 FOR X=1 TO 39
1470  VPOKE X,0
1480 NEXT
1490 RETURN
```

4/ Solution LPB (même machine)

Quelle lenteur désespérante, lorsque le programme ci-dessus en BASIC s'exécute ! On bâille avant d'apercevoir le monstre qui, caché dans un angle mort de votre champ visuel, s'apprête à bondir sur vous !

Nous décidons de convertir le programme ci-dessus en langage LPB. Il nous faut d'abord rechercher dans les tripes logicielles (la ROM) de notre appareil les primitives dont nous avons besoin :

- GOSUB &H9F, qui fournit le prochain caractère tapé au clavier
- GOSUB &H4A, qui lit le contenu de l'écran en HL=X*40+Y
- GOSUB &H4D, qui écrit A dans l'écran en HL=X*40+Y
- GOSUB &H20, qui compare le contenu de HL et de DE

Le résultat en LPB a beaucoup d'air de famille avec l'original BASIC, sauf que les variables que l'on manipule maintenant sont directement les registres du micro-processeur Z80, et que le jeu d'instructions est limité à celles que le micro-processeur sait exécuter directement :

```
1 ORG &HA000:REM (C)1985 CHRISTIAN SCHERER
2 REM        SAVE"scroll.LPB",A:END
10 GOSUB &H9F: REM A=PROCHAIN CARACTERE CLAVIER
20 GOSUB 100
30 GOTO 10
50 PUSH BC,DE,HL:GOSUB &H4A:POP HL,DE,BC:RETURN : REM LIRE VRAM
60 PUSH AF,BC,DE,HL:GOSUB &H4D:POP HL,DE,BC,AF:RETURN : REM ECRIRE VRAM
70 GOTO &H20  : REM COMPARER HL & DE
100 REM TEST FLECHES DIRECTIONNELLES
110 IF A=28 THEN 1100
120 IF A=29 THEN 1200
130 IF A=30 THEN 1300
140 IF A=31 THEN 1400
150 POP HL:RETURN:REM retour au BASIC
1100 REM "vers la droite"
1110 HL=1000
1120 HL=HL-1:FOR B=39
1130 HL=HL-1:GOSUB 50
1140 HL=HL+1:GOSUB 60
1150 HL=HL-1:NEXT
1160 A=" ": GOSUB 60
1170 DE=0:GOSUB 70:IF <> THEN 1120
1180 RETURN
1200 REM "vers la gauche"
1210 HL=0
1220 FOR B=39
1230  HL=HL+1:GOSUB 50
1240  HL=HL-1:GOSUB 60
1250  HL=HL+1
1260 NEXT
1270 A=" ":GOSUB 60:HL=HL+1
1280 DE=1000:GOSUB 70:IF <> THEN 1220
1290 RETURN
1300 REM  "vers le haut"
1310 HL=0
1320 PUSH HL:DE=40:HL=HL+DE:GOSUB 50
1330 POP HL:GOSUB 60
1340  HL=HL+1:DE=960:GOSUB 70:IF<>THEN 1320
1350 A=" ":FOR B=40:GOSUB 60:HL=HL+1:NEXT
1360 RETURN
1400 REM  "vers le bas "
1410 HL=960
1420 HL=HL-1:GOSUB 50
1430 PUSH HL:DE=40:HL=HL+DE:GOSUB 60 :POP HL
1440  DE=0:GOSUB 70:IF<>THEN 1420
1450 A=" ":FOR B=40:GOSUB 60:HL=HL+1:NEXT
1460 RETURN
2000 END 20
9000 DEFUSR=&HA000:PRINTUSR(0):END
9100 OPEN "SCROLL.ASC" FOR OUTPUT AS #1:FOR AD=&HA000 TO &HA0C3:PRINT#1,PEEK(AD
):NEXT:CLOSE 1:END
9200 OPEN "SCROLL.ASC"  FOR INPUT AS #1:FOR AD=&HA000 TO &HA0C3:INPUT#1,A:POKE A
D,A:PRINT PEEK(AD);:NEXT:CLOSE 1:END
```

■

ORGANISATION MATERIELLE

Sa taille réduite (format A5, épaisseur 7 cm) et son poids peu élevé, environ 1 kg, font de FUTURSYS un micro-ordinateur facilement transportable.

Il possède un clavier plat AZERTY, résistant aux chocs et aux agents chimiques, avec des touches incluant jusqu'à quatre fonctions.

L'électronique est du type CMOS. L'écran à cristaux liquides comporte deux lignes de 40 caractères sur un écran virtuel. Ces caractéristiques donnent à FUTURSYS une autonomie pouvant atteindre 24 heures, ceci grâce à une batterie au plomb étanche et sans entretien et acceptant un déchargement complet. L'interface K7 et le port parallèle intégrés permettent l'utilisation de périphériques divers.

Bien que le logiciel FUTURLOG soit implanté té en REPROM, il est possible, sous certaines conditions, de se procurer la dernière version commercialisée de celui-ci. Cette possibilité permet de ne pas être tributaire d'une mémoire de masse après décharge involontaire, par exemple.

ORGANISATION LOGICIELLE

Comme nous l'avons vu dans notre précédent article, la structure de base de FUTURSYS est l'arbre.

Nous allons voir maintenant l'organisation l'organisation générale du logiciel. Deux éléments de connaissance distincts forment les " programmes " élaborés avec FUTURSYS: ce sont les structures et les faits. Il existe des bases (ensembles ordonnés) de structures et des bases (idem) de faits qui sont regroupées, deux par deux, dans des concepts, l'ensemble de ceux-ci constituant la " connaissance " de FUTURSYS. L'intérêt des concepts est d'autoriser des évaluations différentes ou concurrentes dans le même micro-ordinateur.

Décrivons plus en détail ce que sont les structures et les faits.

STRUCTURES

Les structures sont des connaissances élémentaires permettant la transformation en arbre d'une expression initialement sous la forme d'une chaîne de caractères. Une première remarque est que deux bases de structures distinctes pourront donner des arbres différents, donc que deux concepts pourront fournir des décompositions différentes d'une même expression.

Cette transformation en arbre est également appelée analyse de l'expression.

L'arbre résultant de l'analyse comporte des branches de deux type différents, selon qu'elles sont évaluables ou non.

Le fait est qu'en pouvant ainsi analyser une expression, on définit par là même, une syntaxe de l'application, et c'est bien là l'objet des bases de structures.

On voit donc que l'utilisateur a la possibilité de définir sa propre syntaxe, c'est à dire non seulement son vocabulaire, mais également sa grammaire. Nous avons vu que les bases de structures étaient des ensembles ordonnés. Il y a donc, en modifiant cet ordre, la possibilité de modifier l'analyse; ainsi, selon cet ordre, 1+2x3 pourra être compris soit comme (1+2)x3, soit comme 1+(2x3).

Examinons maintenant ce qu'est une structure. Une structure est une chaîne de caractères quelconques hormis quatre d'entre eux qui joueront un rôle particulier lors de l'analyse, en représentant les branches issues du noeud représenté par la structure. Ces quatres caractères sont notés ici: A_u A_F n_u n_F

En effet, analyser une expression va consister à comparer cette expression aux structures considérées comme des masques. Cette opération sera recommencée, s'il y a lieu, sur les sous-expressions correspondant aux branches (aux caractères spéciaux), à l'exception de n_u , n_F (n comme non-analysable).

L'ordre défini au niveau de la base de structures élimine également les litiges pouvant survenir quand plusieurs structures peuvent s'appliquer avec succès à une expression ou sous-expression; ce sera la structure de plus faible rang qui sera choisie.

Ainsi, A_u + A_u , A_uxA_u (dans cet ordre) transforme 1+2x3 en:

```
    +
   / \
  1   x
     / \
    2   3
```

Par contre, la même expression, mais avec, cette fois-ci, A_uxA_u , A_u+A_u , donne:

```
      x
     / \
    +   3
   / \
  1   2
```

Le nombre de caractères spéciaux dans une structure (nombre de branches) est quelconque (Ø à n).

Ainsi, les chaînes: Papa, gateau, f(A_u), A_u A_u+, (A_u), A_u(A_u), A_u:A_u ,A_u!, "n_F",EA_u, somme de A_u et de A_u, SiA_uAlorsA_FSinonA_F, sont-elles toutes des structures potentielles.

Dans la désignation des caractères spéciaux, le rôle du suffixe, "u" ou "F", est de signaler à FUTURSYS que la branche correspondante doit ("u"), ou ne doit pas ("F") être évaluée.

L'intérêt est, dans les structures conditionnelles par exemple, de ne pas évaluer les conséquences tant que la condition n'a pas été elle-même évaluée.

Donc l'expression SiIl.pleutAlorsJe.sors.mon.parapluieSinonRien, et suivant la structure:

SiA_uAlorsA_FSinonA_F

la seule sous-expression évaluée sera Il.pleut.

Nous verrons dans l'article suivant comment interviendront les faits et nous examinerons en particulier les notions d'unification, inférence, algorithmique, combinatoire. ■